

An Easier Version...

What if all of the sizes we are working with are *relatively small* integers? For example, if we could fit 10 pounds and:

Object A is 2 pounds and worth \$40

Object B is 3 pounds and worth \$50

Object C is 1 pound and worth \$100

Object D is 5 pounds and worth \$95

Object E is 3 pounds and worth \$30

We can use dynamic programming!

The solution...

$$\begin{aligned}w_A &= 2 & v_A &= \$40 \\w_B &= 3 & v_B &= \$50 \\w_C &= 1 & v_C &= \$100 \\w_D &= 5 & v_D &= \$95 \\w_E &= 3 & v_E &= \$30\end{aligned}$$

Items

	A	B	C	D	E
1	\$0	\$0	\$100	\$100	\$100
2	\$40	\$40	\$100	\$100	\$100
3	\$40	\$50	\$140	\$140	\$140
4	\$40	\$50	\$150	\$150	\$150
5	\$40	\$90	\$150	\$150	\$150
6	\$40	\$90	\$190	\$195	\$195
7	\$40	\$90	\$190	\$195	\$195
8	\$40	\$90	\$190	\$235	\$235
9	\$40	\$90	\$190	\$245	\$245
10	\$40	\$90	\$190	\$245	\$245

Weight

The hardest situation...

What if our problem just isn't so neat?

$$w_A = 2 \quad v_A = \$40$$

$$w_B = \pi \quad v_B = \$50$$

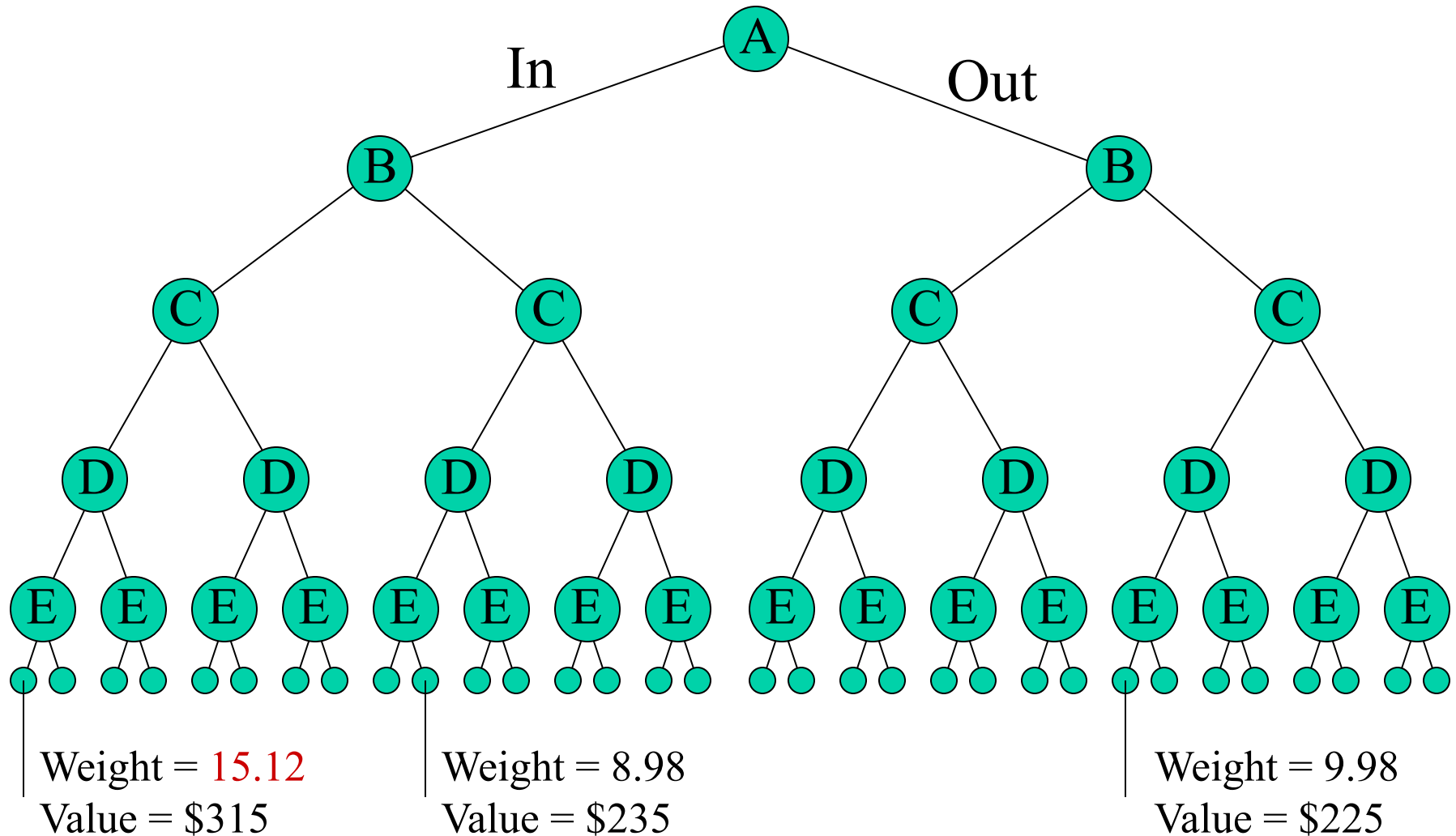
$$w_C = 1.98 \quad v_C = \$100$$

$$w_D = 5 \quad v_D = \$95$$

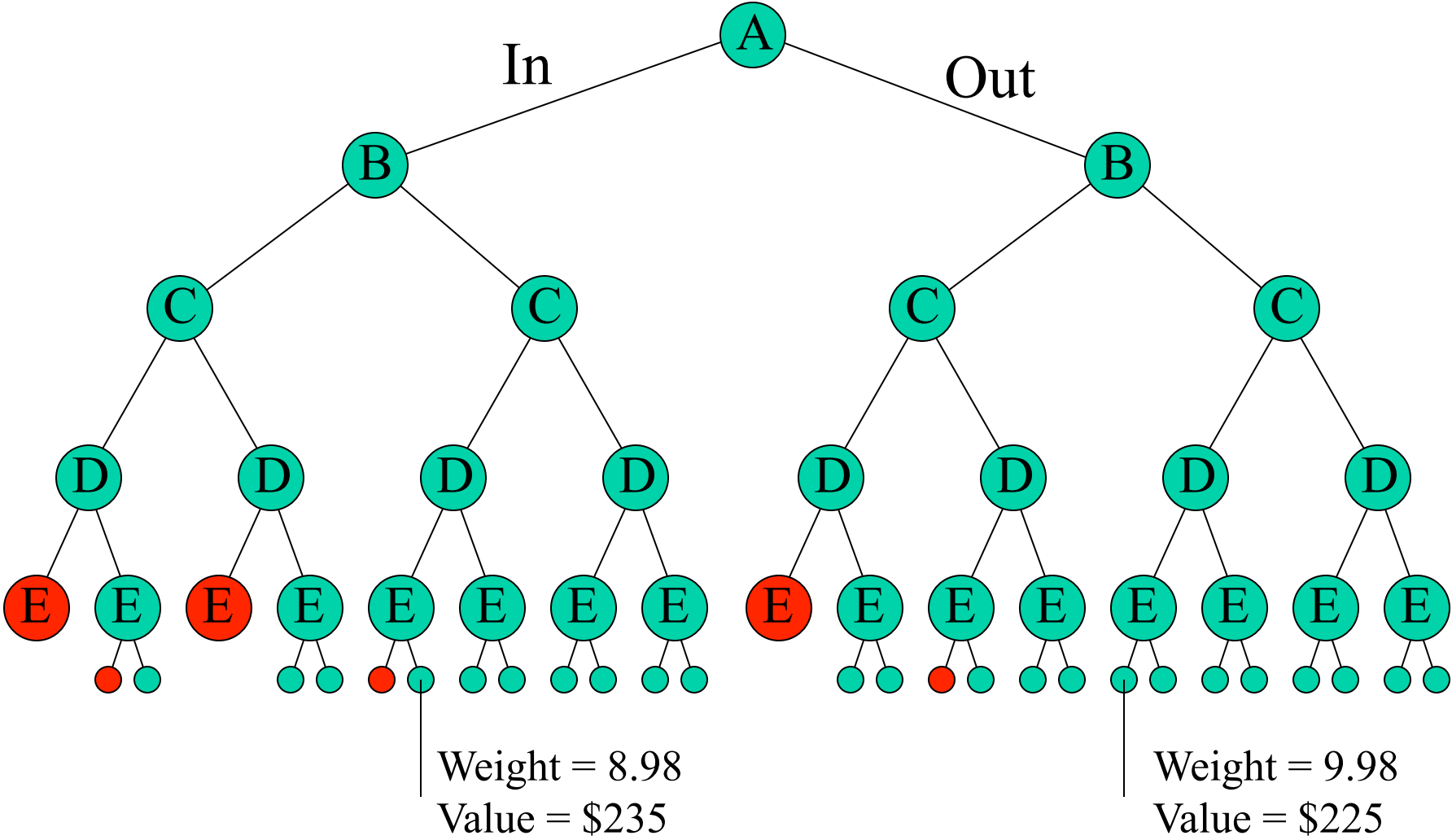
$$w_E = 3 \quad v_E = \$30$$

We have to resort to brute force....

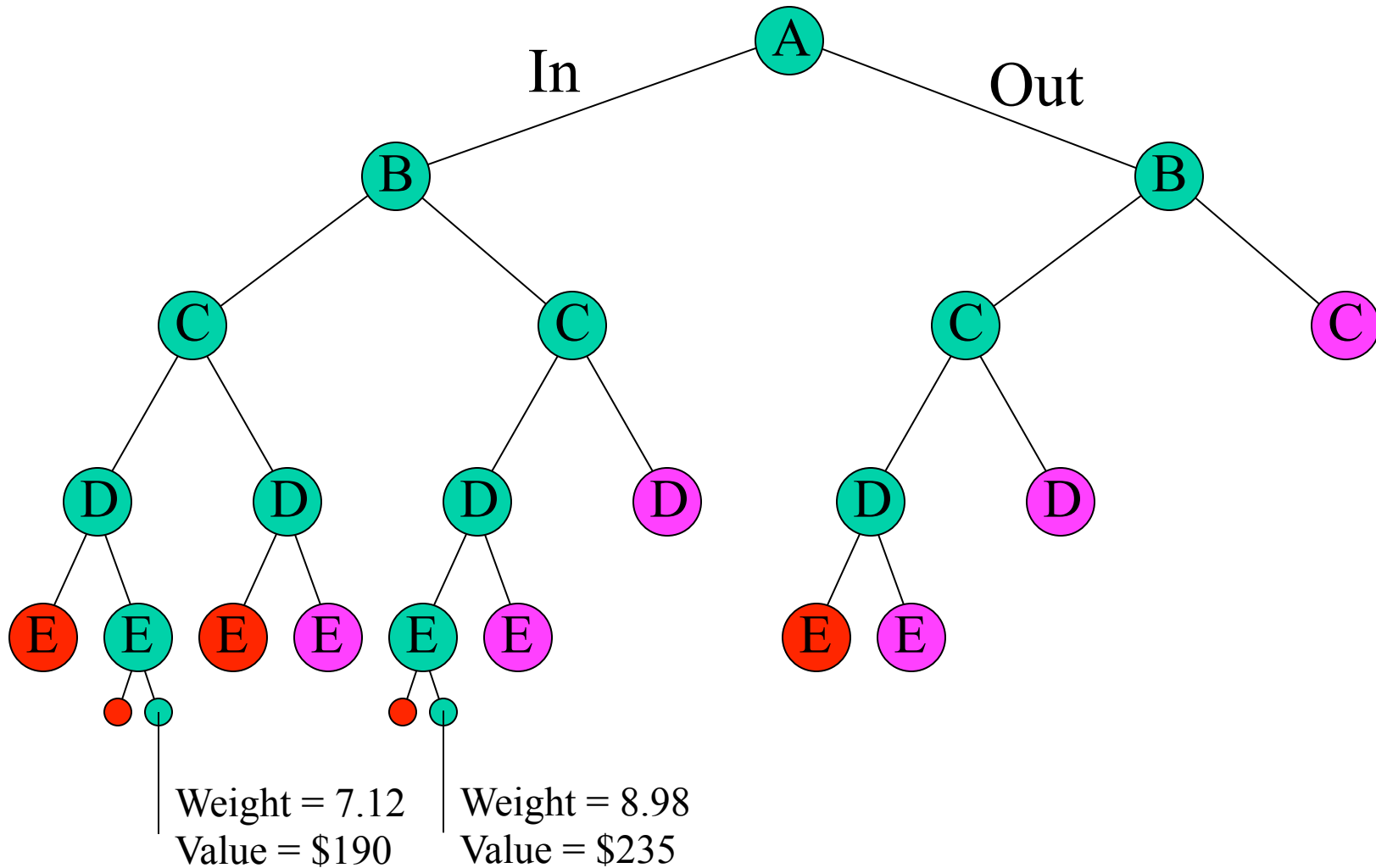
Brute Force: Branching



Brute Force: Backtracking



Brute Force: Branch and Bound



Can we do better than this?

Since the knapsack problem has no inherent order to its breakdown, re-ordering the items can actually improve our performance. This is true of many problems you would use branch-and-bound on.

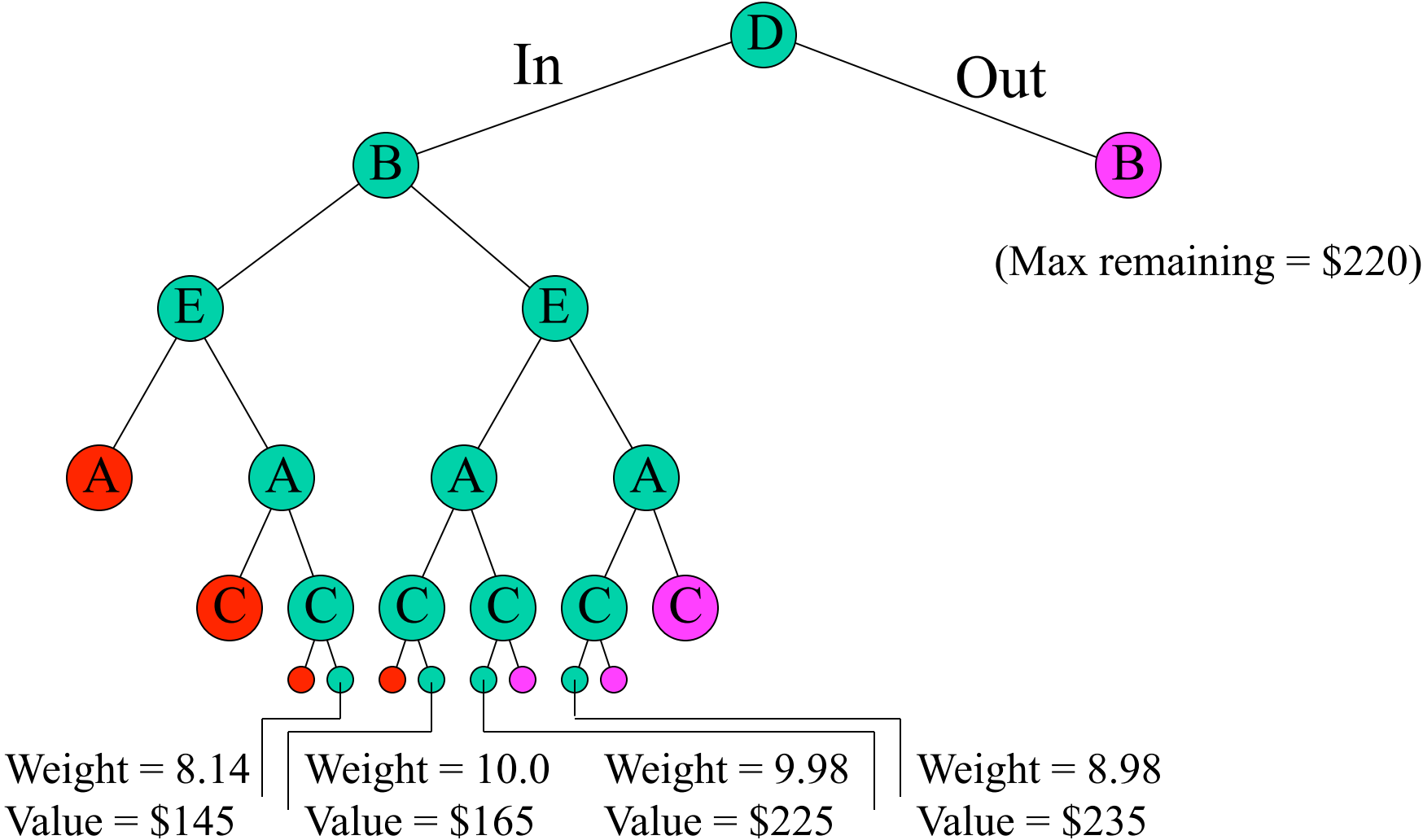
Should we put the heaviest items on top?

- this might cut off sub-trees sooner!

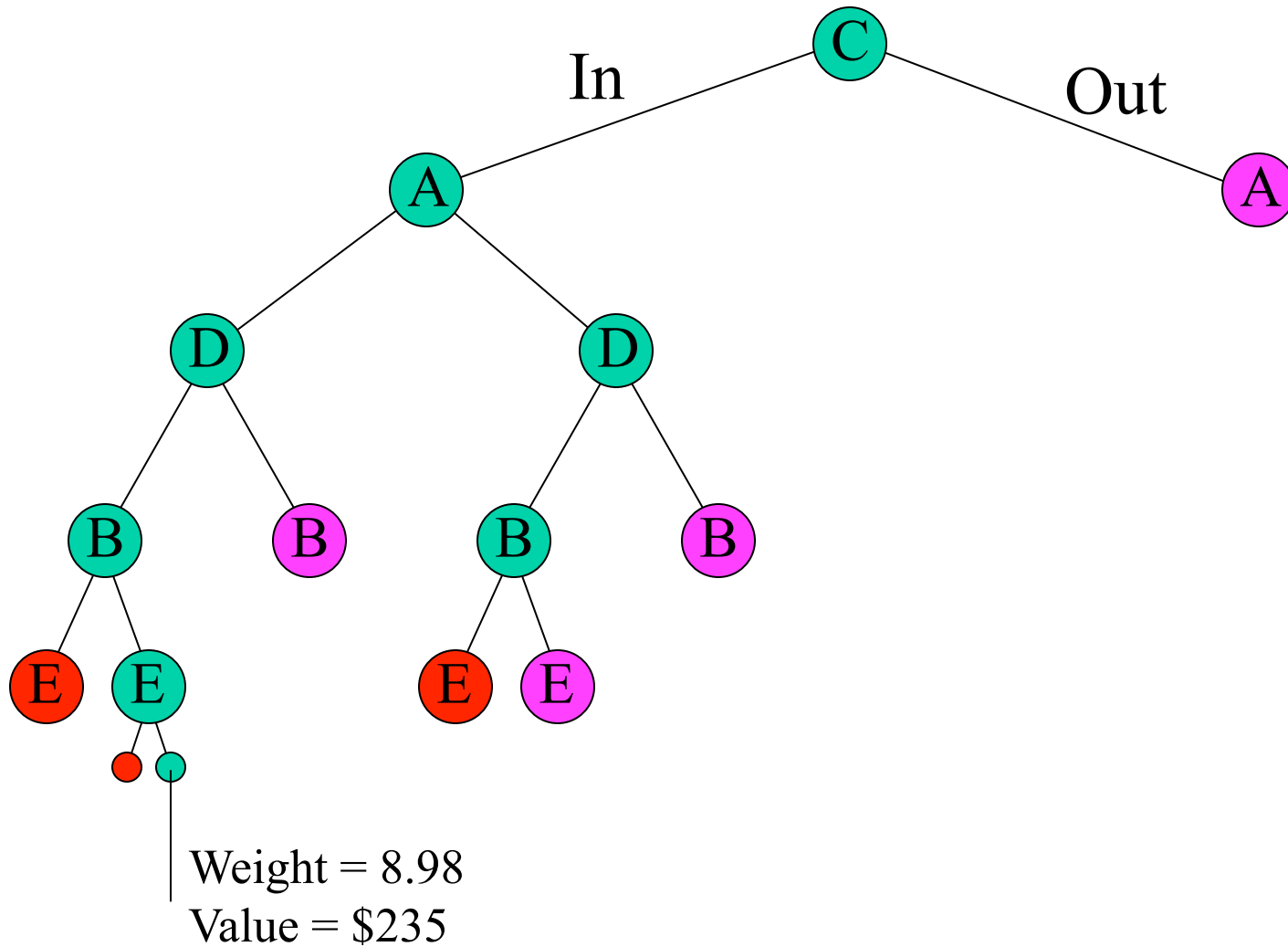
Should we put most valuable/pound items on top?

- this might give us a *good* solution sooner!

Heaviest on Top



Best Value/Weight on Top



Multi-Constraint Knapsacks...

Can we still use techniques like Dynamic Programming when we have multiple constraints to consider?

Example: Mike just bought a new computer and wants to load it up with software. He has \$200 to spend, his computer has 500 gig of hard disk space, and he can only carry 10 lbs. with him on his bicycle. How can he maximize the value of the software he purchases?

MS Office is \$200, takes 52 gig, weighs 2 lbs, and its value is “3”

Quake 7 is \$45, takes 12 gig, weighs 1 lb, and its value is “9”

Product i costs c_i , takes s_i space, weighs w_i , and is valued at v_i .

Can Eight Pieces Cover a Chess Board?

Consider the 8 main pieces in chess (king, queen, two rooks, two bishops, two knights). Can they be positioned on a chessboard such that every square is threatened?

The board on the right (covering 63 sites) was proposed as the closest possible answer in 1849, but could not be proven until computers were used to attack the problem by brute force.

			N	B			R
			N				
R							
	B						
		Q			K		

How many positions to test?

Total brute force:

$$64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57 = 1.8 \times 10^{14}$$

Assuming a computer can verify 1000 arrangements a second, this would take about 6 millennia.

Lets limit things... bishops only have 32 position, and knights and rooks are interchangeable... 1.1×10^{13}

Remove symmetries... 1.4×10^{12} (less than 60 years!)

Backtracking

If we place the pieces in a “bad” order (that is, such that they have the maximum ability to trip over each other), after placing just four pieces we will already be able to start ruling some situations as impossible.

By the time we’re up to 5, many will be impossible, and by 6 (probably just the Knights left to place) almost all would have been ruled out.

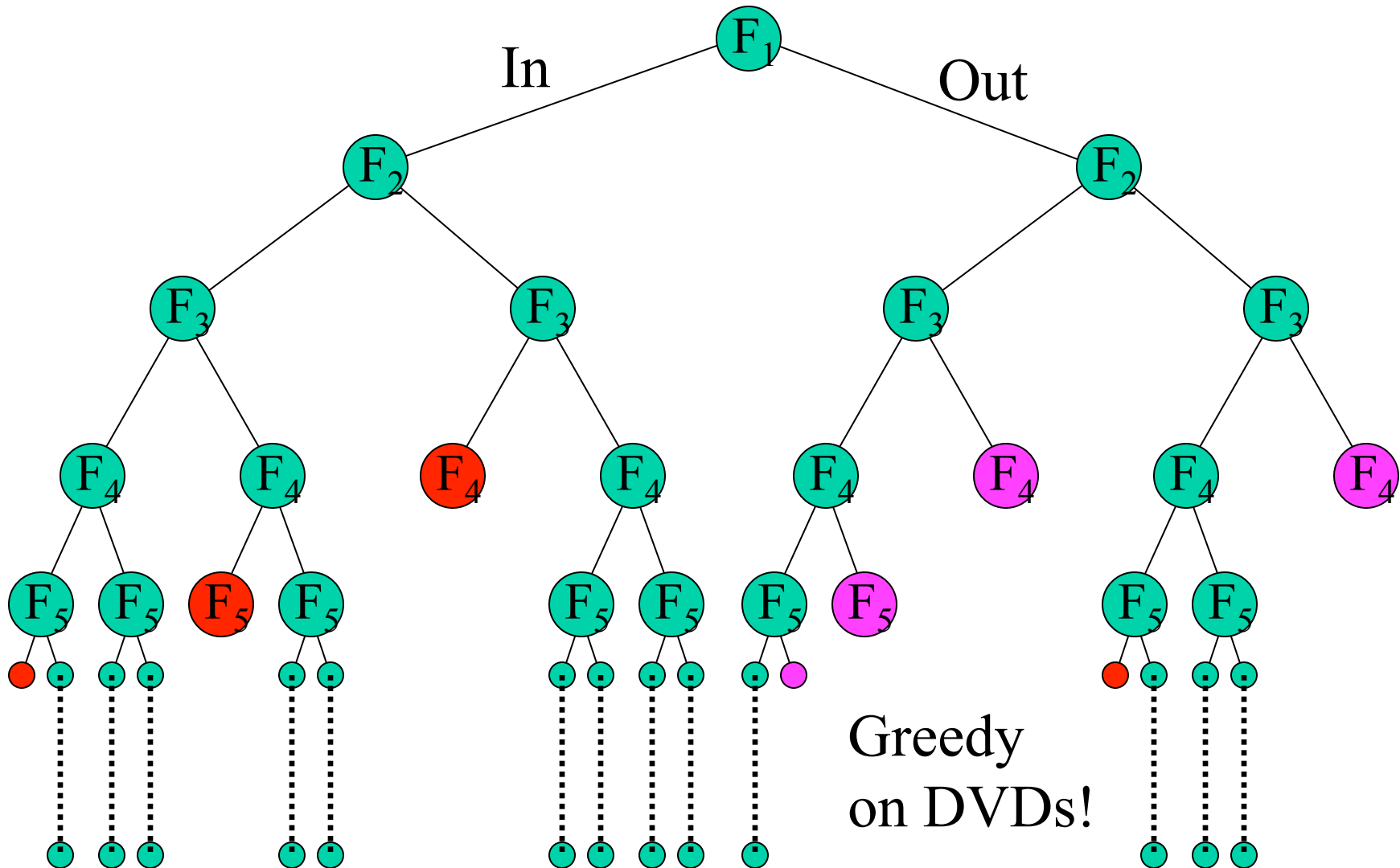
If we only had 6 to place, we only need to consider 8.6×10^8 possibilities -- solvable in only 10 days!

Combining Techniques...

Often we can't use some techniques because *some* of the inputs make it invalid. In these case, we might want to separate out easy instances into a case of their own.

Example: A spy breaks into a mega-corporation and needs to maximize the value of the information he steals. About half of the information is archived on DVDs (all the same size) and the other half is stored on paper in filing boxes of all different shapes and sizes. Given that each piece of information has a fixed value, and he has fixed space to carry stuff, what should he take?

The Breakdown...



But is it any better?

When we divide up a problem like this, we still need to do back-tracking on $n/2$ values. How much of a help does this really give. Plus, the greedy algorithm has to take at least $O(n)$ time for each endpoint!

So, the time our combined algorithm is $O(n \cdot 2^{n/2})$ as opposed to $O(2^n)$ for pure branching.

Lets assume we have a total of $n=50$ items and we have a fast computer that can test 1,000,000 potential solutions every second.

The results...

Using the pure branching, we could have about 2^{50} potential solutions to test. On our fast computer, this would take us over 35 years to finish.

The combo algorithm generates $50 \cdot 2^{25}$ potential solutions, which would take under half an hour. In practice, only 2^{25} solutions are actually generated (the greedy algorithm takes $O(n)$ time at the beginning to generate all potential results and we store them) -- under a minute!

Other combos are possible...

Wallace and Gromit went to a cheese store to buy as much yummy cheese as possible. Unfortunately, they had to carry everything home and could only lift 20 lbs.

The prepackaged cheeses were all an exact number of ounces, while the cheeses packaged at the store were listed to many decimal places. Which cheeses should they buy?

Wensleydale

weight = 5 oz, value = 72

Green Cheese

weight = 4.976342 oz, value = 99

...

